Eurographics Symposium on Parallel Graphics and Visualization (2008)
J. Favre, K. - L. Ma, and D. Weiskopf (Editors)

Parallel Simplification of Large Meshes on PC Clusters

Hua Xiong[1], Xiaohong Jiang[2], Yaping Zhang[1], and Jiaoying Shi[1]

[1]State Key Lab of CAD&CG, Zhejiang University, China
[2]Colledge of Computer Science, Zhejiang University, China

Abstract

Large meshes are becoming commonplace with the advance of 3D scanning, scientific simulation and CAD technology. While there are many algorithms proposed to simplify these large meshes, the time of simplification process is usually very long, especially for those algorithms based on iterative edge collapse. To address this problem, we propose two parallel schemes to speed up simplifying large meshes on a PC cluster. The first parallel simplification scheme partitions a large mesh into small sub-meshes, simplifies these sub-meshes in parallel in an in-core way and finally stitches the simplified versions together. The second scheme generates multiple mesh streams, applies stream simplification to them in parallel in an out-of-core way, and composes the final simplified mesh streams. We have implemented these two parallel simplification schemes and the experimental results show that our methods are able to speed up the iterative simplification of large meshes by a factor of 8 to 19 on a cluster of 24 PCs.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Graphics Systems]: Distributed/network graphics I.3.6 [Methodology and Techniques]: Graphics data structures and data types

1. Introduction

With the advance of 3D scanning, scientific simulation and CAD technology, large meshes containing over 10 million primitives are becoming commonplace [IL05] [CMRS03] [YSG05]. These large meshes can display details of the original models, making them valuable in many applications such as scientific visualization, life science, bioinformatics, culture heritage, city planning, virtual museum, and etc. Limited by the main memory capacity, the CPU and GPU performance of common PCs, however, processing and rendering these large meshes have great challenges. Many acceleration and optimization techniques have been proposed to interactively explore these massive data sets, including parallel rendering, out-of-core approaches, image-based rendering, visibility culling, mesh simplification, multiresolution techniques, mesh compression, mesh layout optimization, and etc. Mesh simplification and multiresolution techniques are two of the most efficient approaches to achieve interactive rendering speed. But previous algorithms usually require rather long pre-processing time for large meshes, even constructing only one simplified version of the input mesh. Speeding up the pre-processing of mesh simplification and multiresolution construction becomes an important topic. This not only benefits downstream mesh processing applications, like editing, texturing, and visualization, but also makes system debugging more convenient.

In this paper, we present two parallel schemes to speed up simplifying large meshes on a PC cluster. These two parallel schemes are based on the mesh cutting and the mesh stream processing concepts respectively. As will be introduced in the related work section, many simplification algorithms for massive meshes are based on the mesh cutting and the stream proceesing concepts. Our parallel schemes and implementations will be a good extension to these existing algorithms. Besides, we also propose some techniques to improve the quality and the efficiency of parallel simplification, including resource management, dynamic task management, and data locality optimization. The main contributions of this paper are followings:

(1) We propose a parallel simplification scheme for large meshes based on the mesh cutting concept. The input mesh is partitioned into many small pieces at first, i.e. the sub-meshes, using a graph partition based algorithm. All sub-meshes are then distributed to the available PCs in a cluster and are simplified in parallel in an in-core way. The simplified sub-meshes are gathered during the run-time and finally are stitched together.

(2) We propose a parallel simplification scheme for large meshes based on the mesh stream processing concept. We generate multiple mesh streams of the input mesh by sorting the mesh primitives and adaptively subdividing its bounding volume. These streams are then distributed to the cluster PCs and simplified in parallel in an out-of-core way. The simplified mesh streams are finally composed together.

(3) To enable balanced parallel simplification, we present a benchmark based resource management scheme and a dynamic task management scheme. We also present a technique to process the boundaries caused by mesh cutting and space subdividing. This technique facilitates the stitching process which is usually unavoidable for parallel massive mesh simplification.

2. Related work

With 3D meshes becoming too large to fit into the main memory, many out-of-core algorithms have been proposed to operate on these large data sets, such as out-of-core simplification, out-of-core construction of multiresolution representations, out-of-core rendering, out-of-core remeshing, and etc. Currently the main approaches to simplify large meshes can be classified as: cutting the mesh into small pieces, designing external memory data structures, and processing the mesh in a streaming or a batch way.

Mesh cutting approaches partition the input large mesh into many small pieces called sub-meshes, each of which can be totally simplified in the main memory. This kind of approaches is perfectly suitable for large meshes with regular spatial distribution of primitives such as terrain models [Hop98]. For other general large meshes, space subdivision based mesh cutting [Pri00] or graph partition based mesh segmentation [YSG05] may be employed. Boundaries among partitioned sub-meshes are usually simplified in a hierarchical manner by using a different partition in each iteration. But for meshes with very irregular primitive distribution, like some heavily folded CAD models, mesh cutting methods usually result many isolated sub-meshes and boundary vertices [Sha06], boundary maintenance and stitching will cost a lot of time.

Approaches based on external memory data structures usually build an out-of-core representation for indexing global primitives of the input large mesh and also a compact in-core skeleton of the representation. A mapping between the out-of-core representation and the in-core skeleton is designed carefully to achieve efficient data management. During the simplification, these approaches only load the needed geometric primitives into the main memory through either an explicit or an implicit data scheduling scheme [DP02] [CE97]. Based on this concept, simplification methods using iterative edge collapse [CMRS03] and vertex clustering [SG05] have been implemented.

Batch processing approaches usually work on the triangle soup representation. The triangle soup is generally simplified through multiple passes of out-of-core scanning and sorting of primitives. A simplification algorithm combining this concept with the vertex clustering method has been implemented [Lin00]. Furthermore, streaming simplification methods require the primitives of the input mesh is streamable, i.e. with good data locality of the primitives with respect to their storage positions. This property can be obtained by many ways, including spatial sorting [WK03], using streaming mesh representation, i.e. interleaving vertices and triangles with vertices finalization information [ILGS03], and etc. By using an in-core buffer and scanning the streamable mesh primitives, the whole input mesh can be simplified progressively without being totally loaded into the main memory.

3. Cutting based parallel simplification

The first parallel simplification scheme we proposed is based on the mesh cutting concept. The general process consists of three steps: (1) mesh cutting, i.e. cutting the input large mesh into small sub-meshes each of which can be loaded and simplified in the main memory. (2) parallel simplification, i.e. distributing the partitioned sub-meshes to the cluster PCs and simplifying them in parallel using iterative edge collapse operator. (3) sub-meshes stitching, i.e. gathering the simplified sub-meshes and merging them together. The basic idea of this approach, i.e. cutting-simplifying-stitching, is straightforward and has been implemented in a serial way by many researchers. But to efficiently extend this idea to a parallel environment with distributed memory, each step has some problems to be attacked. We will present these problems and our solutions in following sections.

3.1. Mesh cutting

Many mesh partition algorithms have been put forward to help digital geometry processing for large meshes [Sha06]. One kind of methods is to subdivide the bounding volume of the mesh with planes, such as a uniform grid, an octree, and etc. Another kind of methods is to segment the mesh surface based on its primitive connectivity. The partition results of these two kinds of methods are mainly affected by the geometry and the topology of the mesh respectively.

In order to enable balanced parallel simplification for all PCs and control the simplification quality of the sub-meshes, the partitioned sub-meshes should be nearly equally sized in term of the triangle count. Apparently, the space subdivision methods are hard to meet this requirement even if they proceed recursively. Besides, the boundary vertices and triangles should be as few as possible. This desirable property not only helps improve the quality of the final simplified mesh but also facilitates the stitching process. Considering all above constraints, we adapt the cluster decomposition method based on graph partition [YSG05]. The steps of the mesh cutting algorithm are:

(1) Compute the bounding box of the mesh by scanning the vertex list. Subdivide the bounding box with a uniform grid (with dimension of 2^nx2^nx2^n). Scan the vertex list to compute the vertex count and the average vertex position in each grid cell. The uniform grid can be kept in the main memory by only storing non-empty cells and hashing these cells with their linear indices in the grid as the hash keys.

(2) Construct a graph whose nodes represent the non-empty cells and are weighted by the vertex count in each cell. Edges are established between a node and its k-nearest neighboring nodes. We use the METIS graph partitioning library [KK98] to recursively decompose the constructed graph into clusters of cells. The METIS library gives nearly uniform cluster size and can minimize the count of edge cuts.

(3) Assign vertices to clusters according to the cell index of each vertex. We use a vertex map file to record the cluster index and the local index in a cluster for each vertex. This map file will help resolve triangles and determine which cluster a triangle belongs to in the next step. All vertices in a cluster are extracted and stored into a separate vertex file.

(4) Partition the triangles into clusters. If all three vertices are in the same cluster, the triangle is assigned to the cluster. Otherwise, it is a boundary triangle crossing two or three clusters. The vertices of the boundary triangles are marked as boundary vertices in the vertex map file. Edges connecting with a boundary vertex are not eligible for edge collapse. All triangles in a cluster are also extracted and stored into a separate triangle file. This file is merged with the vertex file into the final sub-mesh file. Because boundary vertices will not be removed during the simplificaton of a sub-mesh, we keep them at the beginning of the sub-mesh file. However, all boundary triangles (with their vertex records) are stored in a separate file to help stitch sub-meshes.

The mesh cutting result of the Thai Statue model is given in Figure 1. The mesh cutting step usually costs only a few percentage of the whole process time and is not a bottleneck of the system, so we do not parallelize it.

3.2. Balanced parallel simplification

The mesh cutting makes the parallel simplification viable. Each sub-mesh is small enough to be simplified in the main memory using iterative edge collapse and quadric error metrics [GH97]. Two problems in the parallel simplification process may affect the performance greatly. The first one is the resource management of a PC cluster, especially for those heterogeneous PCs clusters whose nodes may have quite different main memory capacity and CPU performance. The second one is the simplification task management. Only balanced task distribution and execution can fully exploit the power of the PCs cluster. To address these problems, we propose a benchmark based resource management scheme and a dynamic task distribution scheme.

Figure 1: *Mesh cutting result of the Thai Statue model. The resolution of the uniform grid is 512x512x512 (n=8). The count of clusters is 512.*

3.2.1. Resource management

Knowing the performance of each PC before parallel simplification is important to achieve workload balancing. We use a benchmark mesh to evaluate the performance of each PC. This benchmark test constructs related data structure for simplifcation and performs half edge collapse till all triangles are removed. We think the simplification throughput (the ratio of the triangle count to the simplification time) can indicate the PC's performance. Using this performance parameter, a straightforward static method for task distribution is to partition sub-meshes into sets which have approximate estimated simplification time. However, the run-time performance may result rather different simplification time, especially in a Grid environment whose node may run multiple tasks simultaneously. Instead, we propose a dynamic task management scheme and use the performance parameter to control the size of the input buffer for each PC.

We use a controller PC to monitor the status of each PC and to allocate and release the PC resource during the simplification course. Each PC reports its status by sending a 'heart beating' message to the controller over a period of time. If a PC breaks down, its unfinished tasks will be taken back amd re-assigned to other available PCs by the controller. This resource management scheme is easy to extended to a Grid environment.

3.2.2. Dynamic task management

Based on the resource management scheme, we present a dynamic task management scheme to achieve more balanced parallel simplification than the static task distribution method. During the simplification course, each PC dynamically applies for sub-meshes from the controller and returns the simplified versions to it. To prevent the cluster PCs

from being idle, each PC maintains two buffers, i.e. the input buffer and the output buffer, to cache the sub-meshes before and after simplification. If the occupancy ratio of the input buffer falls below a threshold (we use 0.8 in our experiments), the PC applies for a batch of sub-meshes to renew the input buffer. The simplified sub-meshes are inserted into the output buffer and are returned to the controller while the output buffer is not empty. The PC performance parameter is used to limit the maximum size of the input buffer as that all PCs will finish the simplification at the same time if all the input buffers are full. These two kinds of buffers also help hide the data transmission latency. This dynamic task management scheme is illustrated in Figure 2.

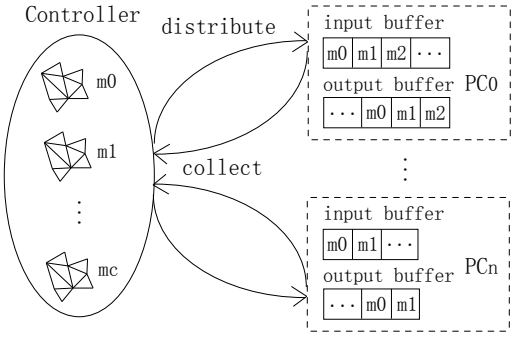

Figure 2: *The dynamic task management scheme for balanced parallel simplification. The controller dynamically distributes sub-meshes to the input buffers of cluster PCs. When a sub-mesh is simplified, it is inserted into the output buffer and is collected by the controller. Please note that the input buffer size of a PC is determined by its performance.*

3.3. Stitching and post-processing

When all sub-meshes are simplified and are returned to the controller, they are stitched together and the boundary is simplified further. As we have described before, boundary vertices are marked in each sub-mesh and boundary triangles are stored in a separate file. This technique greatly facilitates the stitching process.

By scanning the vertex list of each simplified sub-mesh, the vertices are merged together directly. The starting index and the range of vertices of each sub-mesh are also recorded. We then scan the triangle lists, change their vertex indices to the new ones and merge them. Finally, we add the boundary triangles to the final simplified mesh. Because the boundary vertices are keep at the beginning of their sub-mesh file, their relative order does not change during the simplification process. It is easy to compute the new vertex indices of boundary triangles from their vertex records, i.e. the cluster index and the local index. Figure 3 gives an illustration of the data structure for boundary stitching.

Because all the edges connecting with boundary vertices are disabled to collapse, the final simplified mesh may have non-uniform distribution of triangles, especially along the boundaries of sub-meshes. To solve this problem, we apply another pass of simplification to those edges connecting with boundary vertices only. And its simplification ratio is the same as that of the sub-meshes. The final simplified mesh is usually small and thus the boundary simplification can be performed totally in the main memory.

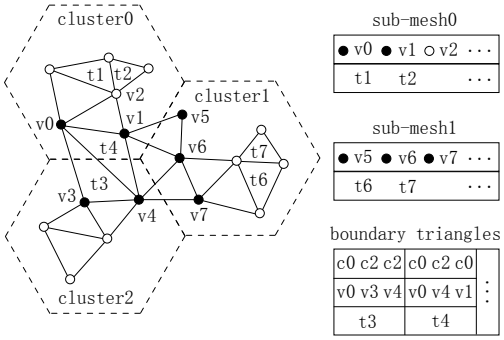

Figure 3: *Illustration of boundary stitching. Triangles spanning two or three clusters are boundary triangles and their referenced vertices are marked as boundary vertices, shown as filled circles. Boundary vertices are stored at the head of the vertex lists. Boundary triangles are stored separately with records of vertices' cluster indices and local indices.*

3.4. Experimental results

We present the experimental results of the cutting based parallel simplification in this section including the parallel simplification performance and the task management efficiency. Our test environment is a cluster of 24 PCs. Each PC has two 2.4GHz CPUs and 1GB of RAM. All PCs are connected by a Gigabit Ethernet.

3.4.1. Parallel simplification performance

To evaluate the performance of the mesh cutting based parallel simplification approach, we tested our system on all 24 PCs and compared the results with the simplification performance on a single PC. As shown in Table 1, the maximum speedup of our system is 19. We used the edge collapse method as in [Pri00] to simplify sub-meshes. The stand-alone PC implementation also operated on the same set of sub-meshes, but simplified them one by one. Compared with the result of [WK03], in which they reported about 33 minutes to simplify the David 1mm model of 56M triangles to 1 percentage, it cost our stand-alone PC implementation 81 minutes to simplify the Lucy model to 1 percentage. However we can simplify this model in 6 minutes using 24 PCs.

Table 1: *Results of the mesh cutting based parallel simplification approach. The task distribution time is included in the partition time and the time of gathering simplified sub-meshes is included in the stitching time.*

Meshes	Thai Statue	Lucy
#Triangle in	10,000,000	28,055,742
#Triangle out	200,052	280,102
Percentage (%)	2	1
#Sub-mesh	512	1024
Partition	0:00:41	0:03:26
Simplification	0:00:32	0:01:31
Stitching	0:00:16	0:00:45
Total time	0:01:29	0:05:42
Single PC time	0:27:45	0:80:24
Speedup	19:1	14:1

3.4.2. Task management efficiency

By changing the count of sub-meshes, we tested the simplification efficiency of our dynamic task management scheme and compared it with the static scheme. We kept the primitive count of each sub-mesh invariant and randomly added some sub-meshes from the initially partitioned sub-mesh set. We used a heterogeneous PC cluster. 16 of the 24 PCs were kept. The remaining 8 PCs were replaced by PCs of one 800MHz CPUs and 512MB of RAM. As show in Figure 4, with the increase of the count of sub-meshes, the simplification time difference between the dynamic and the static task management schemes became more obvious. This can be explained as the static scheme tends to inaccurately estimate the workload of task sets. The experimental result also shows that our scheme has good scalability with respect to the mesh size. But for the static scheme, its performance declines greatly when the count of sub-meshes increases.

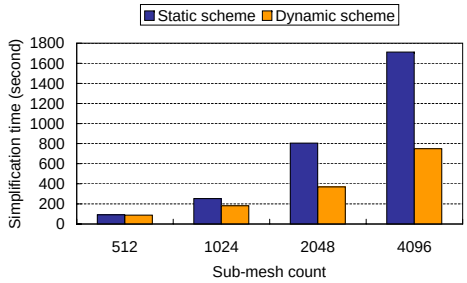

Figure 4: *Comparison of task management efficiency. Our dynamic task management scheme shows better performance to drive balanced parallel simplification.*

4. Stream based parallel simplification

Recently, many stream based methods have been put forward to facilitate simplifying large meshes [IL05] [ILGS03] [WK03]. These methods progressively load a batch of primitives (including vertices and triangles) into the main memory, simplify them and output the simplified primitives to a disk file. Compared with the mesh cutting based approaches, they do not require the stitching step and can simplify a large mesh by scanning its disk file once. Our second parallel simplification scheme is based on this kind of methods. The general process includes three steps: (1) mesh stream generation, i.e. creating multiple streams of the input mesh each of which is suitable for stream simplification on a PC. (2) parallel stream simplification, i.e. distributing the mesh streams to the cluster PCs and performing simplification on all streams in parallel. (3) stream composition and post-processing, i.e. collecting simplified mesh streams, composing the final simplified mesh and processing the boundaries further.

4.1. Mesh stream generation

One prerequisite of parallel stream simplification is to generate multiple mesh streams for the cluster PCs. Our stream generation method is also based on mesh partition because of its efficiency. But compared with the sub-meshes of the mesh cutting based approach, the count of mesh streams is much fewer. Because each stream can totally reside on the disk and can be simplified by sequentially scanning. This is enabled by improving their data locality. One may partition the mesh using the graph partition algorithm as introduced before and then improve the data locality of each sub-mesh individually. We employ another method which optimizes the data locality of the input mesh first. The streams can be simply generated by space subdividing the bounding volume of the optimized mesh.

4.1.1. Data locality optimization

An important concept for large meshes processing, especially for those out-of-core approaches is data locality, i.e. the temporal and spatial proximity of primitives access. Accessing a large mesh with poor data locality usually results lots of cache misses and degrades run-time performance [NBS06] [YLPM05]. For those large isosurfaces generated by the marching cube method and those massive meshes obtained by sequential 3D scanning, their primitives are usually built up layer by layer resulting good data locality. Some methods have been put forward to optimize mesh data locality, including geometrical sorting, topological sorting, space filling curves, spectral sequencing, and etc.

We adopt the geometrical sorting for its simplicity. The vertices are sorted along the longest coordinate axis of the bounding box. An index map is used to record the vertex indices before sorting and to update the triangle list. Triangles are sorted using its smallest vertex index as the sort key. For extremely large meshes, out-of-core method can help this process as in [Lin00]. Figure 5 shows the result of the data locality optimization.

Figure 5: *Mesh data locality optimization. Shading color displays the triangle sequence before (left) and after (right) the geometrical sorting.*

Figure 6: *Stream generation result based on the data locality optimization and the adaptive space subdivision for the Lucy model. The stream count is 24.*

4.1.2. Adaptive space subdivision

The geometrical sorting approach introduces a natural way to generate multiple mesh streams, i.e. space subdividing the bounding box of the input mesh. To achieve balanced parallel stream simplification, we employ an adaptive space subdivision according to the available PCs in a cluster. The adaptive space subdivision proceeds in following steps:

(1) Given the count of the available PCs n, partition the triangles into n batches with equal number of triangles in each batch. In fact, this is equivalent to subdividing the mesh along the longest coordinate axis of its bounding box.

2) For each batch of triangles, compute the smallest and the largest vertex index, extract vertices between this range from the vertex file, increase their reference counters, and update the triangles' vertex indices.

3) Merge the vertices and triangles into a mesh stream. Denote those vertices whose reference counter is not equal to 1 as boundary vertices and triangles referencing them as boundary triangles. Usually, the reference counter of a boundary vertex is 2 because the proximity of space subdivision regions.

The stream generation result of Lucy model is given in Figure 6. While this method is easy to implement, it is not suitable for large meshes with very irregular primitive distribution. We think the way of mesh stream generation should depend on the type of the application and the input mesh.

4.2. Parallel stream simplification

For the generated streams, each cluster PC performs streaming simplification using an algorithm similar to the stream decimation algorithm proposed by Wu and Kobbelt [WK03]. The main difference of our algorithm is that we adopt the indexed triangle format instead of the triangle soup. So it does not need to reconstruct the mesh connectivity through hashing vertices with their coordinates. Once the one-ring neighbors of a vertex are all identified, edges connecting with this vertex but not connecting with any boundary vertices are eligible for collapse. We assume that the simplified sub-mesh can be totally kept in the in-core buffer. So only the INPUT and DECIMATION operations for the stream are actually needed. This simplifies the streaming simplification process.

4.3. Stream composition and post-processing

When all the cluster PCs finish simplification of the assigned mesh streams, they return the simplified versions to the controller PC for stream composition. The composition process is similar to the stitching process of the mesh cutting based approach. But it is simpler because the places where the mesh streams connect are spatially ordered. And with this constraint, the composition process can be also parallelized by using the parallel image composition concept from the parallel rendering literature. Similarly, boundaries are finally simplified to remove those densely tessellated areas of the merged mesh.

4.4. Experimental results

We present the experimental results of the stream based parallel simplification approach in this section. We implemented the stream simplification algorithm proposed in [WK03]. It cost us about 26 minutes to simplify the Lucy model to 1 percentage on a single PC. Compared with their

Table 2: *Results of the mesh stream based parallel simplification approach. The task distribution time is included in the stream generation time and the time of gathering and stitching streams is included in the composition time.*

Meshes	Thai Statue	Lucy
#Triangle in	10,000,000	28,055,742
#Triangle out	200,013	280,172
Percentage (%)	2	1
#Streams	24	24
Generation	0:00:32	0:02:12
Simplification	0:00:12	0:00:48
Composition	0:00:08	0:00:25
Total time	0:00:52	0:03:25
Single PC time	0:08:20	0:25:45
Speedup	9:1	8:1

results, our approach obtained a speedup factor of 8, as shown in Table 2. All the PCs with two 2.4GHz CPUs and 1GB of RAM were used.

5. Discussion and analysis

For the cutting based parallel simplification scheme, we do not parallelize the mesh cutting step. This is because the graph partition based method is of high run-time efficiency and is not a bottleneck in our experiments. But for extremely large meshes, the cutting step may affect the overall system performance. For this case, we can first partition the input mesh into several large blocks in space and execute the proposed mesh cutting algorithm for each block in parallel.

Similarly, if the sub-meshes stitching step is heavily loaded, we can also parallelize this step. This is because the simplified sub-meshes are returned during the simplification course. We need to maintain a neighbor table and the status of neighbors for each cluster. Once a neighboring cluster is simplified and is ready for merging, the controller PC sends these two sub-meshes to available cluster PC for stitching. We think many parallel image composition algorithms from the parallel rendering literature will be very helpful to reduce the network transmission and the overall stitching time.

For the stream based parallel simplification scheme, the data locality improvement step may be a bottleneck for extremely massive meshes. If adopting the geometric sorting method, one can modify the serial algorithm to a parallel version easily as followings: (1) cutting the input mesh into blocks. (2) out-of-core sorting each block in parallel. (3) executing parallel merge sort for each pair of blocks. While for other methods such as topological sorting and spectral methods, it is non-trivial to find an efficient parallel solution. We think this may be a potential limitation of this scheme.

6. Conclusion and future work

We have presented two parallel simplification approaches based on the mesh cutting and the mesh stream processing concept respectively. For the first scheme, we have presented how to partition the input mesh into sub-meshes with balanced primitive count, how to perform dynamic task distribution and parallel simplification on a PC cluster, and the technique to stitch the simplified sub-meshes together. For the second scheme, we have presented methods of generating multiple mesh streams and performing simplification on these streams in parallel. We have implemented these two schemes and the experimental results have shown a speedup factor of 8 to 19 on a cluster of 24 PCs.

We expect to experiment with our system with more massive geometric data sets and to further improve its performance. There are some avenues for future work. The GPU processing ability has been improved quickly. Combining with its parallel processing units [DT07] and even assembling a GPU cluster, interactive simplification of large meshes might be obtained. Multiple mesh streams generation is still worthy of further study. Recently, many spectral methods for mesh processing and analysis have been proposed [ZvKD07]. It may give us good heuristics for this purpose. Besides, construction of a multiresolution representation for a large mesh usually costs much pre-processing time. We hope our work can foster research in this direction.

7. Acknowledgements

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312105 and the NSFC project of "Digital Olympic Museum" under Grant No.60533080. We would like to thank the Stanford Graphics Group for providing the data sets.

References

[CE97] COX M., ELLSWORTH D.: Application-controlled demand paging for out-of-core visualization. In *Proceedings of IEEE Visualization 1997* (1997), pp. 235–244.

[CMRS03] CIGNONI P., MONTANI C., ROCCHINI C., SCOPIGNO R.: External memory management and simplification of huge meshes. *IEEE Transactions on Visualization and Computer Graphics 9*, 4 (2003), 525–537.

[DP02] DECORO C., PAJAROLA R.: Xfastmesh: Fast view-dependent meshing from external memory. In *IEEE Visualization* (2002), pp. 363–370.

[DT07] DECORO C., TATARCHUK N.: Real-time mesh simplification using the gpu. In *Symposium on Interactive 3D Graphics (I3D)* (Apr. 2007), vol. 2007, p. 6.

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of ACM SIGGRAPH 1997* (1997), pp. 209–216.

[Hop98] HOPPE H.: Smooth view-dependent level-ofdetail control and its application to terrain rendering. In *IEEE Visualization 1998* (1998), pp. 35–42.

[IL05] ISENBURG M., LINDSTROM P.: Streaming meshes. In *Proceedings of IEEE Visualization 2005* (2005), pp. 231ÍC–238.

[ILGS03] ISENBURG M., LINDSTROM P., GUMHOLD S., SNOEYINK J.: Large mesh simplification using processing sequences. In *Proceedings of IEEE Visualization 2003* (2003), pp. 465–472.

[KK98] KARPIS G., KUMAR V.: A multiresolution representation for massive meshes. *Journal of Parallel and Distributed Computing 48*, 1 (1998), 96–129.

[Lin00] LINDSTROM P.: Out-of-core simplification of large polygonal models. In *Proceedings of ACM SIGGRAPH 2000* (2000), pp. 259–262.

[NBS06] NEHAB D., BARCZAK J., SANDER P. V.: Triangle order optimization for graphics hardware computation culling. In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), pp. 207–211.

[Pri00] PRINCE C.: *Progressive Meshes for Large Models of Arbitrary Topology*. Master's thesis, University of Washington, 2000.

[SG05] SHAFFER E., GARLAND M.: A multiresolution representation for massive meshes. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (2005), 139–148.

[Sha06] SHAMIR A.: Segmentation and shape extraction of 3d boundary meshes. In *State-of-the-art Report, Eurographics* (2006), pp. 137–149.

[WK03] WU J., KOBBELT L.: A stream algorithm for the decimation of massive meshes. In *Proceedings of Graphics Interface 2003* (2003), pp. 185–192.

[YLPM05] YOON S.-E., LINDSTROM P., PASCUCCI V., MANOCHA D.: Cache-oblivious mesh layouts. In *Proceedings of ACM SIGGRAPH 2005* (2005), pp. 886–893.

[YSG05] YOON S.-E., SALOMON B., GAYLE R.: Quick-vdr: Out-of-core view-dependent rendering of gigantic models. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 369–382.

[ZvKD07] ZHANG H., VAN KAICK O., DYER R.: Spectral methods for mesh processing and analysis. In *Proc. of Eurographics State-of-the-art Report* (2007), pp. 1–22.